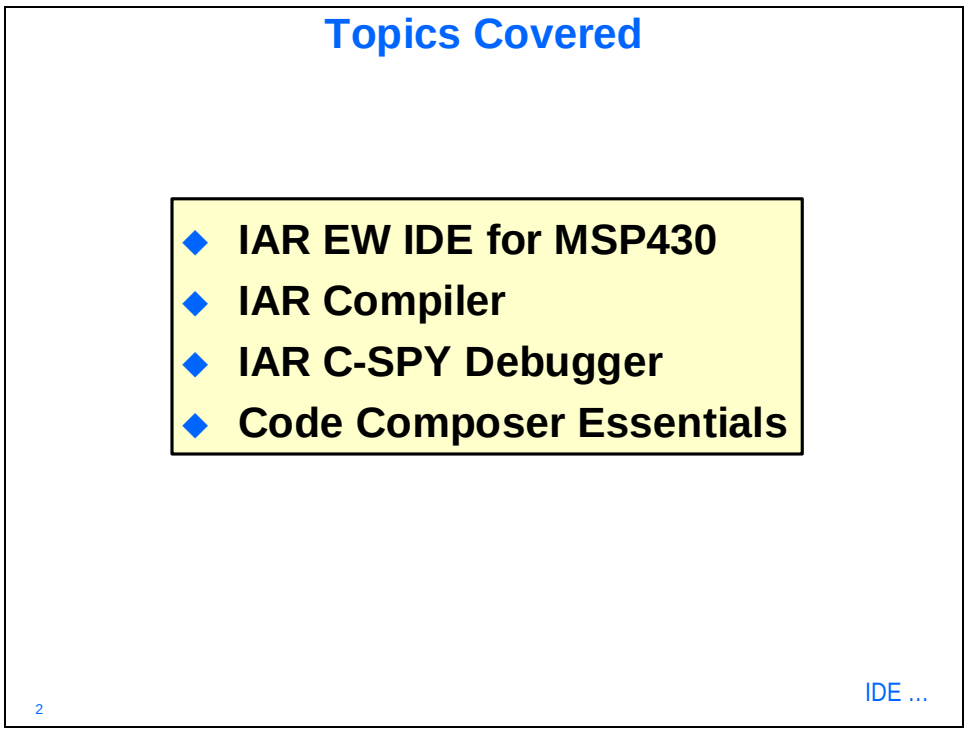
IAR Embedded Workbench

Introduction

For those of you who have worked with IAR Embedded Workbench before, this will be a review. If you have not, this module and lab will give you a quick introduction to some of the features

Learning Objectives



*** enjoy this photograph of a polar bear on a snowfield ***

Module Topics

IAR Embedded Workbench 2-1

Module Topics 2-3

IAR Embedded Workbench 2-5

Workspace Organization 2-6

Compiler 2-7

Debugger 2-8

Code Composer Studio 4.0 2-9

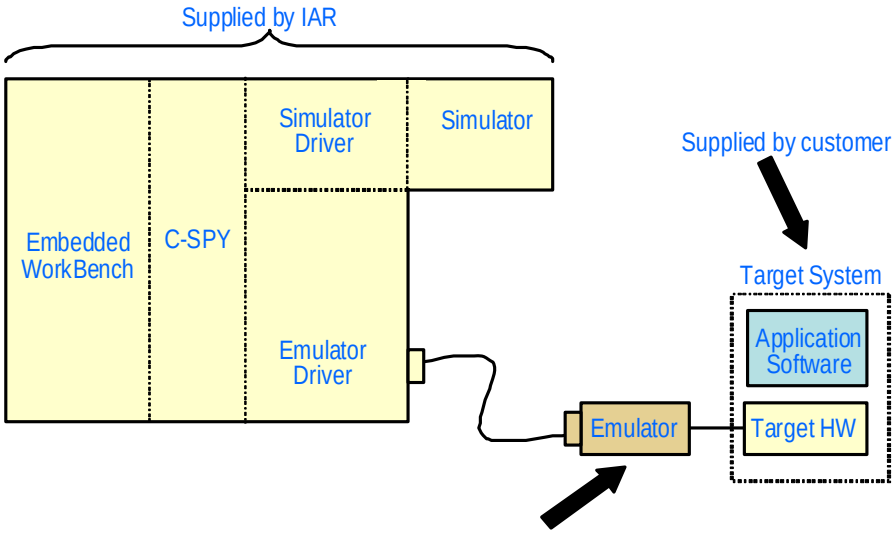
Lab 2 – Exploring IAR Embedded Workbench 2-11
Description: 2-11
Hardware list: 2-12
Software list: 2-12

Procedure 2-13
Setting up IAR Embedded Workbench 2-13

*** this page has little to offer ***

IAR Embedded Workbench

IAR Tools and Target System



Some workbench highlights ...

4

Easy-to-use IDE

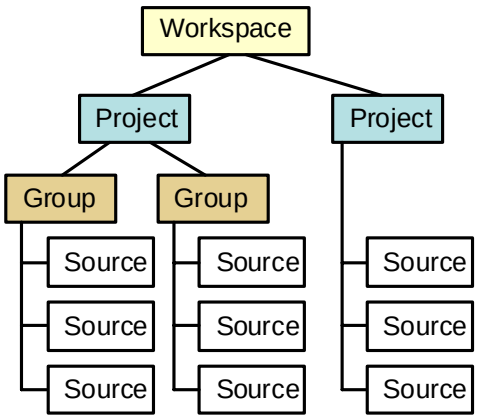
- **Powerful Project Manager can arrange multiple projects in a workspace**
- **Smart Source Browser - Easy navigation to functions, types, variables and macros and with filtering possibilities**
- **Tons of context-sensitive help**
- **Open architecture allows easy expansion of the standard tool chain**
  - External editor
  - Source code control system integration

Workspace organization ...

5

Workspace Organization

Workspace Organization

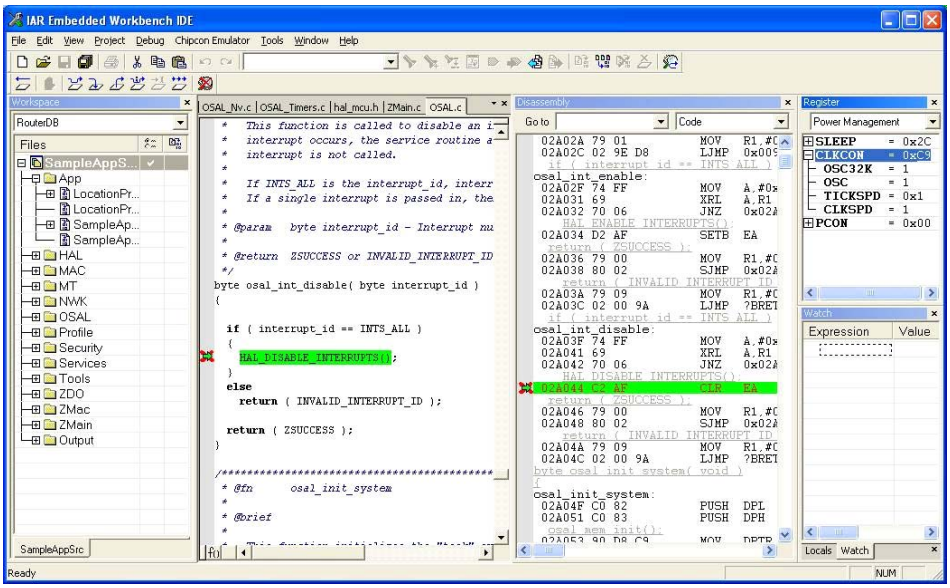


- Workspaces can contain multiple projects.
- Projects contain sources and groups. Default settings for the project can be inherited.
- Groups contain sources, other groups, and can override any inherited settings.
- Configurations can easily switch between different builds

A look at the screen ...

6

IDE Screenshot



C Compiler ...

7

Compiler

IAR C Compiler

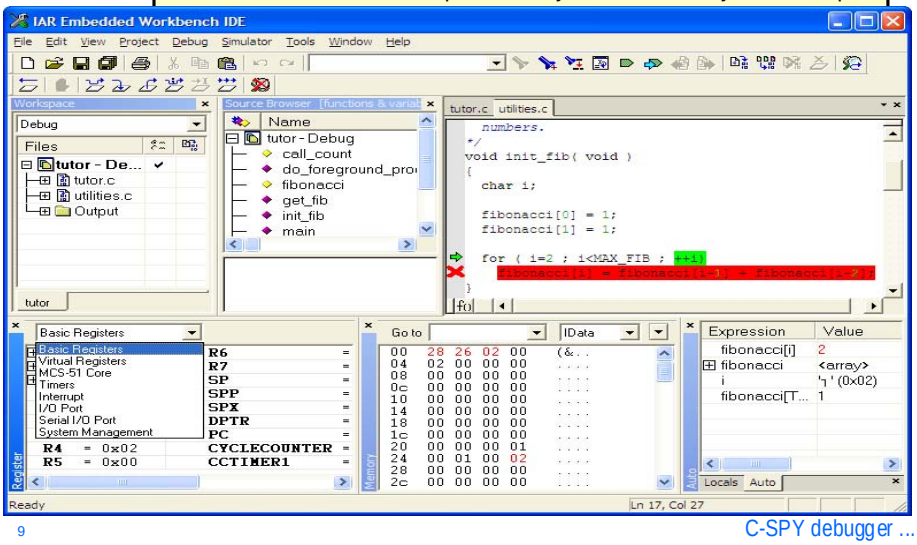
- Highly optimized ISO/ANSI standard C compiler
- Customizable C library, pre-built and in source form
- Comprehensive language extensions
- Flexible segment commands allow detailed control of code and data placement
- Multiple levels of optimizations for code size and execution speed
- Major functions of the optimizer can be controlled individually, for example loop unrolling
- MCU specific optimizations

8

Highlights ...

IAR Embedded WorkBench Highlights

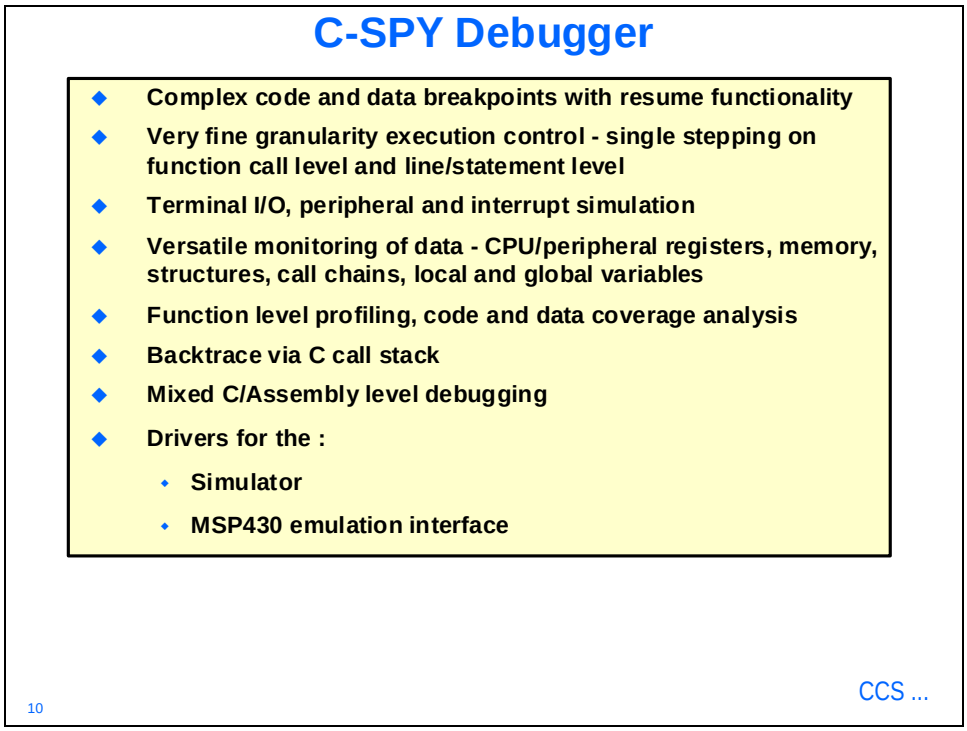
- Ready-made I/O register definition files
- Comprehensive documentation with efficient coding hints
- Context-sensitive help with library function and keyword lookup



9

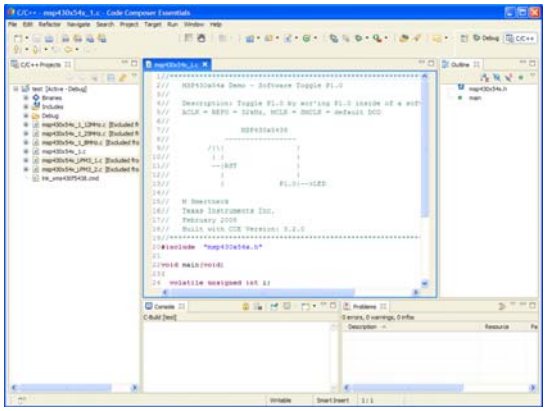
C-SPY debugger ...

Debugger


C-SPY Debugger
Complex code and data breakpoints with resume functionality
Very fine granularity execution control - single stepping on function call level and line/statement level
Terminal I/O, peripheral and interrupt simulation
Versatile monitoring of data - CPU/peripheral registers, memory, structures, call chains, local and global variables
Function level profiling, code and data coverage analysis
Backtrace via C call stack
Mixed C/Assembly level debugging
Drivers for the :
Simulator
MSP430 emulation interface
10
CCS ...

Code Composer Studio 4.0

Code Composer Studio 4.0



16K version:
Free
MSP430 Full Version:
$499

- MSP430 C compiler, assembler and linker
- Source Code Debugger
- Integrated Visual Project Manager
- Hardware and virtual breakpoints
- Eclipse Integrated editor

Lab Time ...

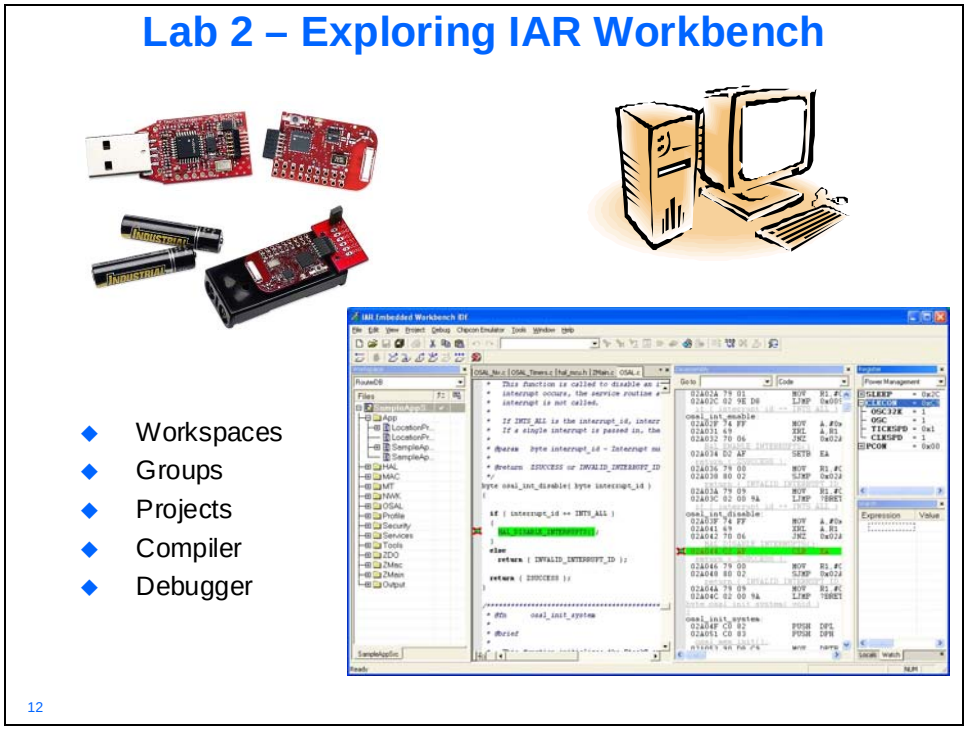
11

*** yet another blankity-blank page ***

Lab 2 – Exploring IAR Embedded Workbench

Description:

IAR Embedded Workbench is one of the most popular IDEs on the market today. You can't go wrong learning to use this popular and capable tool.



Hardware list:

- ✓ 3 eZ430-RF2500 Target Boards
- ✓ 2 Battery Modules
- ✓ 4 AAA Batteries
- ✓ 1 eZ430-RF2500 Emulator Board
- ✓ 1 USB Extender Cable

Software list:

- ✓ IAR Embedded Workbench for MSP430 version 4.11D

(You will find shortcuts for the above application on the desktop)

Procedure

Setting up IAR Embedded Workbench

1. Double-check your hardware

Hopefully, none of the other sneaky people in your class has fooled with your hardware setup. Make sure the USB extension cable, EZ430-RF2500 emulator and target board are all connected properly.

2. **Start the tool**

Double-click on the **IAR Embedded Workbench** shortcut on the desktop. This starts the MSP430 version of the tool.

When the startup window appears, click the **Create new project in current workspace** button. Other than creating the original **main.c** source file, we're going to do this from scratch. In the next window, the **Tool chain** should be **MSP430**, then click **OK**. When the **Save in** window appears, Navigate to:

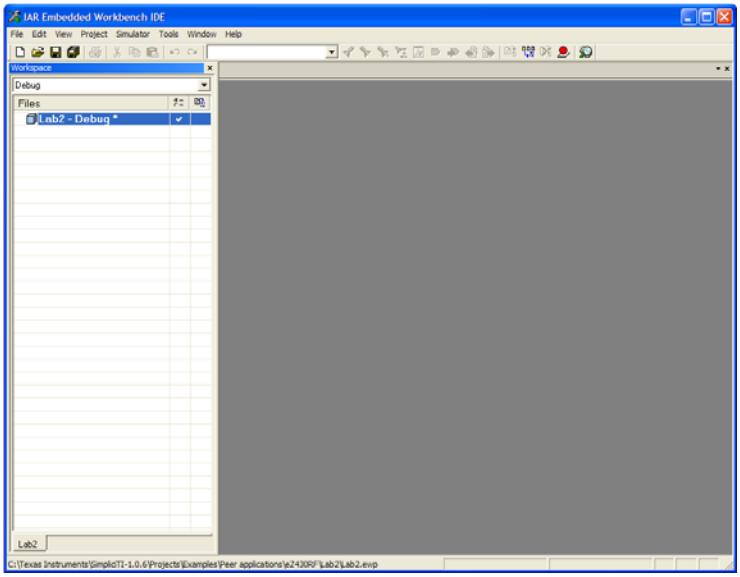
C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\Peer applications\ eZ430RF\Lab2

Name the project file **Lab2** and click **Save**.

We're going to make use of some the SimpliciTI features in later labs, so the lab files are located in the SimpliciTI folders.

3. Get familiar with the IDE

Take a look at the Workspace:



The **Menu bar** is on top, the **Workspace** window is on the left and the **Editor** window is on the right.

4. Add Groups

Let's set up a couple of folders so the project parts are organized. **Right click** anywhere in the **Workspace** window and select **Add → Add Group …** Type **Components** into the **Group name** box and click **OK**.

Follow the same procedure to add a group named **Source**.

5. Add BSP files

There's no way I'm wasting my life writing code that interacts with port/pin combinations when I have a Board Support Package at hand, so let's add the BSP file to the project.

Right-click on the Components group and select **Add → Add Files…** Navigate to **C:\Texas Instruments\SimpliciTI-1.0.6\Components\bsp** , select **bsp.c**, **bsp.h** and **bsp_macros.h** all at once and click **Open**.

6. Add main.c to the Project

Add **main.c** from **C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\ Peer applications\eZ430RF\Lab2** to the **Source** group.

7. Open main.c for editing

Double-click on **main.c** in the **Workspace** window. The file will open for editing (with a tab above) in the editor window. Note the **bsp.h** and **bsp_leds.h** files, they add the BSP functionality for the LEDs.

The code is pretty simple … initialize the BSP (always first), turn on the green LED, then toggle both LEDs every 1/3rd second or so. This delay loop is the WRONG way to write MSP430, so don't get used to it. In this case, though, it simplifies the code.

8. Add the Include Paths

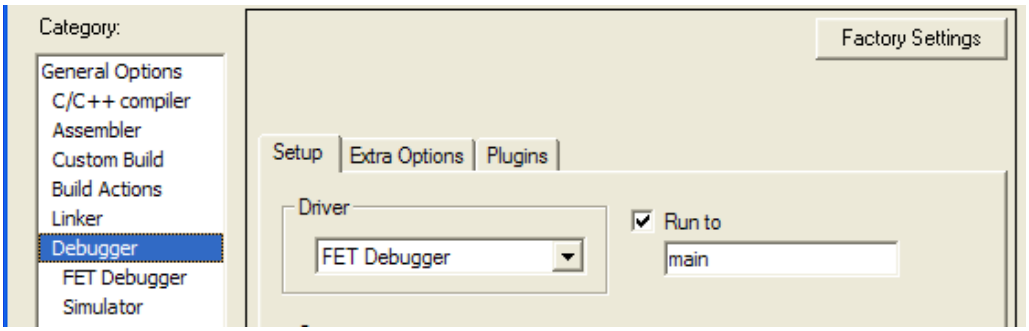
In order for the header files to work, the compiler needs to know the correct paths to these files. Make sure the project name **Lab2 – Debug** is highlighted in the Workspace window (click on it). From the **Menu bar,** click on **Project → Options** and pick the **C/C++ compiler** category. Click on the **Preprocessor** tab and add the following to the **Additional include directories** box. Normally, the easiest way to do this is to use Windows Explorer to navigate to the folder and cut/paste the path. But I've placed a **Paths.txt** file in the **Lab2** folder … you can just cut/paste from that if you like.

C:\Program Files\IAR Systems\Embedded Workbench 5.0\430\inc
C:\Texas Instruments\SimpliciTI-1.0.6\Components\bsp
C:\Texas Instruments\SimpliciTI-1.0.6\Components\bsp\drivers
C:\Texas Instruments\SimpliciTI-1.0.6\Components\bsp\boards\EZ430RF

9. Define the Correct Part and Target

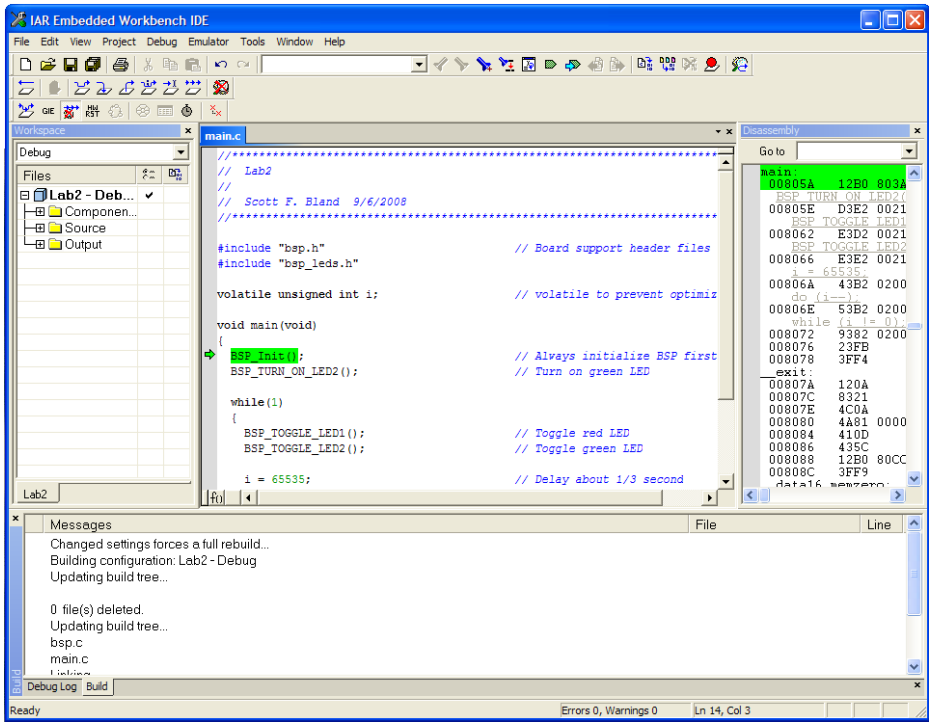
Select **General Option** from the **Category** list and change the **Device** to **MSP430x2xx Family → MSP430F2274**. This is the MCU on the eZ430-RF2500 board.

Select **Debugger** from the **Category** list and then select **FET Debugger** under the **Setup** tab in the **Driver** box. This is the single most common and frustrating mistake people make when setting up an MSP430 project. It can take way too much time to realize that your program is working, but it's loading on the simulator. Also note that **Run to main** is checked; we'll see the advantage of that in a second. Click on the **FET Debugger** category. Under the **Breakpoints tab**, check the **Use software breakpoints** checkbox, The eZ430 emulator provides a single hardware breakpoint, and this will give us many more. Click **OK**.

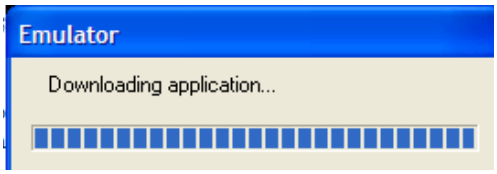


10. Build and Load

Click the **Debug** button on the right side of the **Menu** bar. You'll be prompted to save your workspace. Name it **Lab2** and click **Save**. The project should build properly. If it does not, notify your instructor. A series of build messages will appear at the bottom of you IAR window and the project view will change to the debug view, like below. The Workspace window is still on the left, the **C debug** window is in the center and the **Disassembly** window is on the right. The green highlights and arrow denote the current position of the program counter.



If you were watching closely, you also saw this window pop up, then disappear:



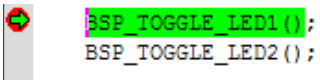
The IAR Compiler/Assembler/Loader has created an executable file from your source code and programmed it into the flash memory of the MSP430. It is now completely non-volatile and could be disconnected and run on the battery. But we have other plans …

11. Run

Find the **Go** button on the menu bar and click it. Note the other run control buttons like **Reset**, **Step Over**, **Step Into**, etc. If everything works the way it's supposed to, the LEDs on the target board should be rapidly flashing.

12. Breakpoints

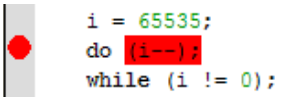
Code execution is trapped in the while(1) loop. So let's set a breakpoint on the first instruction inside it by double-clicking just to the left of the instruction. You can also right-click on the instruction and select Toggle Breakpoint(Code).



Execution should quickly stop at the breakpoint, as shown by the green arrow. Click on the **Go** button a couple of times and watch the LEDs toggle as the loop runs. Feel free to experiment with the other run control button now.

13. Watch Window

Set another breakpoint on the **`do (i--);`** instruction.



Right-click on any line containing the **i** variable and select **Add to Watch**. The **Watch** window should appear on the right side of your IAR screen. At this point, **i should be 0**. Click the **Go** button and watch code execution stop at the new breakpoint. Repeat this procedure a few times.

14. Remove the Breakpoints and Shut down IAR Embedded Workbench

Double-click on both breakpoints (red dots) to remove them. Click on the **Stop Debugging** button to return to the editor window. **Close** IAR Embedded Workbench. If you are prompted to save anything; please do.

15. Run on Batteries

Remove the **target** board from the eZ430 emulator and carefully **connect** it to one of the **battery modules**. It's easy to mis-align the connector, so watch what you are doing. Connect the **power jumper** across the two pins. The LEDs should flash and life should be great. **Disconnect** the jumper and place on one of the pins for safekeeping.

You're done

TEXAS
INSTRUMENTS